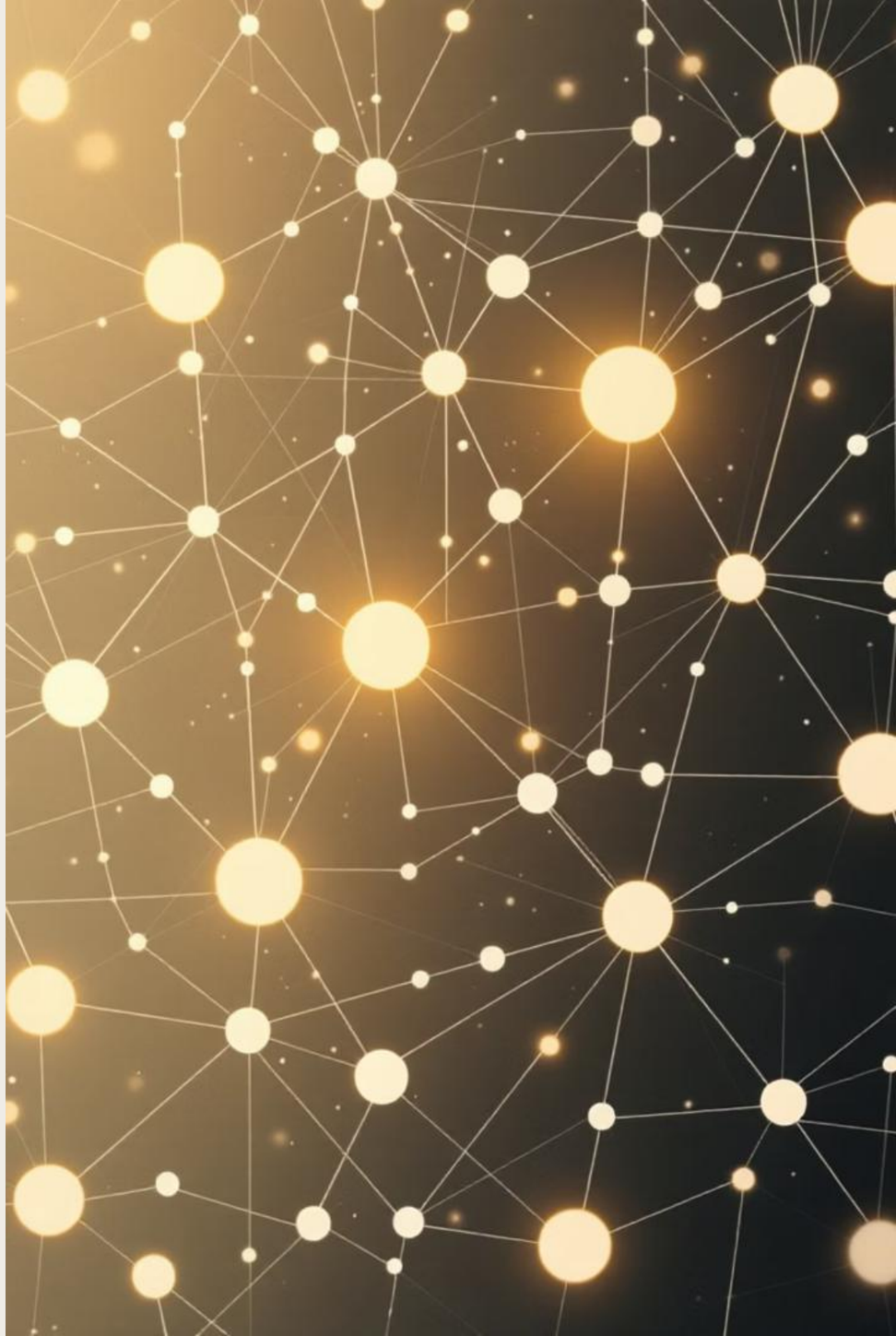
Interwoven Networks: Cross-Domain Community Mining via Layer-Aware NMF

Shaik Masthan Jilani

23075074 CSE B.Tech.,

Indian Institute of Technology (BHU), Varanasi



Index

Motivation & Goal

Datasets & Data Collection

Preprocessing & Alignment (align.py)

Model: Symmetric NMF (shared H, per-layer D)

Optimization & Implementation details

Initialization & Layer Weighting

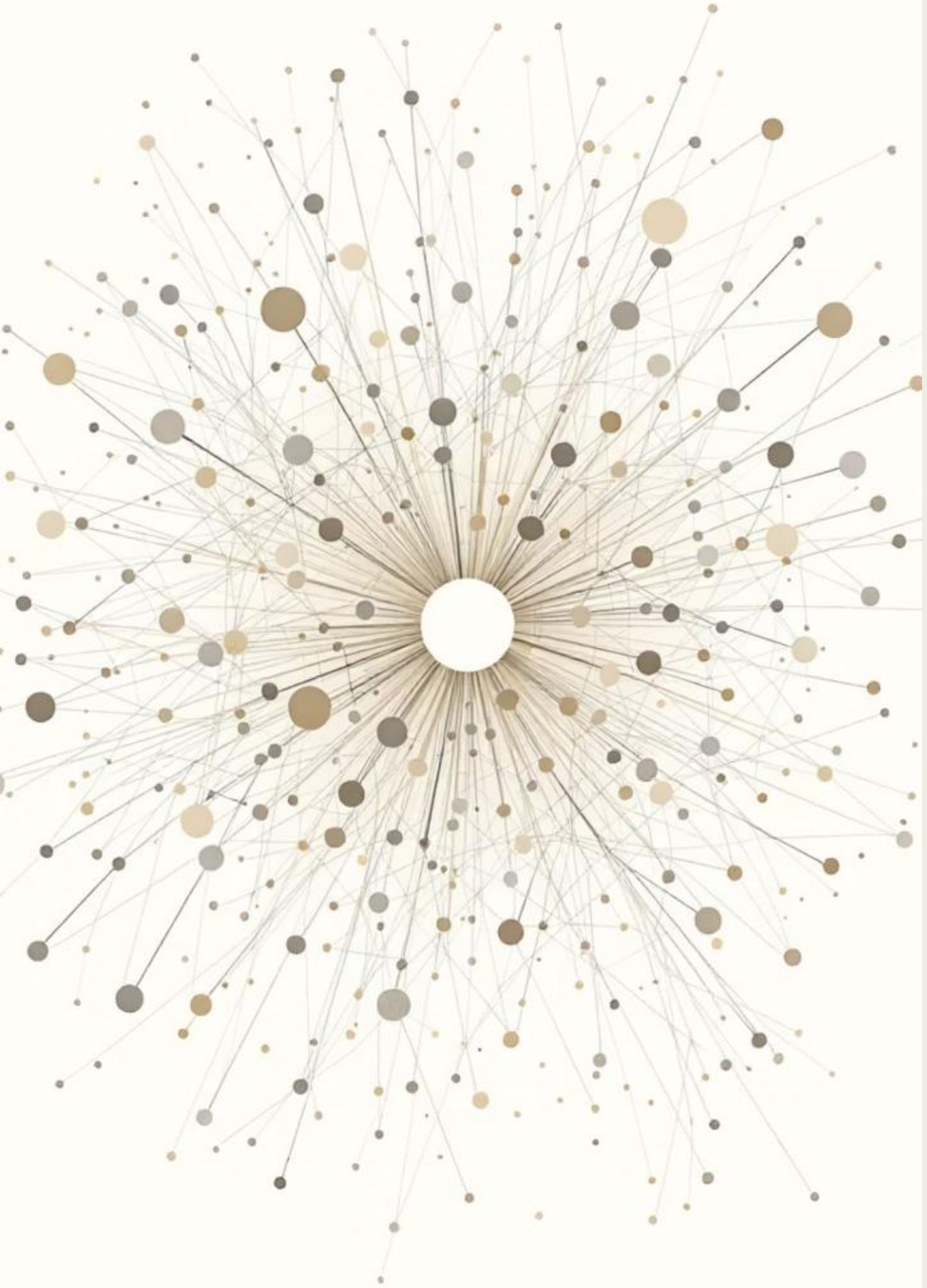
Evaluation metrics

Grid search, consensus & results

Practical issues and fixes

Conclusions & Takeaways

Code & artifacts



Motivation and Goal

Real-world entities interact via multiple relations (email, coauthorship, social networks).

Goal: learn a single, shared community embedding H (n×k) while allowing layer-specific strength via diagonal D^(ℓ).

Advantages: interpretability, cross-layer consistency, compact representation.

Task: implementation of Symmetric NMF, run hyperparameter grid, evaluate with multiple metrics, and form consensus.

Datasets: Sources and diversity

- Enron email graph (communications) — organizational interaction pattern.
- ca-HepPh (coauthorship) — academic collaborations.
- ca-GrQc (coauthorship) — smaller, sparser collaboration graph.
- Optional datasets / recommendations: Orkut, DBLP, LiveJournal for social/academic mixtures.
- Data sources: SNAP, public Enron corpus, DBLP dumps (all the public datasets mwntioned in report).

Preprocessing & Alignment (align.py):

- Read raw edge lists (u v per line), ignore comments and empty lines.
- Choose node alignment mode: UNION (all nodes) or INTERSECTION (only common nodes).
- Map original node ids → contiguous 0..n-1 mapping (save node_mapping.json).
- Build symmetric CSR adjacency matrices, remove self loops, clip multi-edges → {0,1}.
- Save aligned matrices as compressed .npz (scipy.sparse.save_npz).

Model: Symmetric NMF (shared H, per-layer D)

- Objective (weighted): minimize Σ_ℓ w_ℓ || A^(ℓ) - H D^(ℓ) H^T ||_F^2 subject to H≥0, D^(ℓ) diag ≥0
- H ∈ R^{n×k} : shared soft membership (rows: node memberships over k communities).
- D^(ℓ) ∈ R^{k×k} diagonal: scales community importance per layer.
- Interpretation: each layer is explained by same communities but with layer-specific strengths.

$$\min_{H\geq 0,\ D^{(\ell)}\ 0,\ D^{(\ell)}\ \text{diagonal}} \sum_{\ell} w_{\ell} \left\| A^{(\ell)} - H D^{(\ell)} H \quad \right\|_F^2 + \mu \sum_{\ell} \left\| D^{(\ell)} \right\|_F^2 + \lambda \left\| H \right\|_1$$

$$\text{subject to } H \geq 0,\ D^{(\ell)} \geq 0, \quad \forall \ell$$

and updated iteratively as:

$$H \leftarrow H \odot \frac{\sum_{\ell} w_{\ell} A^{(\ell)} H D^{(\ell)}}{\sum_{\ell} w_{\ell} H S^{(\ell)} + \mu_H H + \lambda + \varepsilon}, \quad S^{(\ell)} = D^{(\ell)} (H \quad H) D^{(\ell)}.$$

Optimization & Updates:

- Alternate updates: solve for each $D^{(\ell)}$ via small k×k linear system; multiplicative updates for H.
- Compute D: solve M d = b where $M_{cd} = (h_c^T h_d)^2$ and $b_c = h_c^T A h_c$.
- Update H multiplicatively using numerator = $\Sigma_\ell w_\ell A^{(\ell)} H D^{(\ell)}$, denominator = $\Sigma_\ell w_\ell H S_\ell + \mu H$ ($S_\ell = D M D$).
- Use regularizer μ to stabilize updates and avoid numerically degenerate solutions.
- Avoid forming dense n×n matrices — compute objectives via trace identities to save memory.

Initializations:

- random: H ~ |N(0,1)| + ε (unbiased start)
- louvain: run Louvain on first layer → one-hot H warm start
- reason: nonconvex problem; init affects convergence & stability

Weight methods (w_ℓ):

- none/uniform: w_ℓ = 1
- inv_nnz: w_ℓ ∝ 1 / nnz(A^(ℓ)) (sparse layers boosted)
- sqrt_inv_nnz: w_ℓ ∝ 1 / sqrt(nnz(A)) (milder)
- degree_inv: w_ℓ ∝ 1 / avg_degree
- custom: user-provided weights; normalization to sum L

Evaluation Metrics:

Per-layer modularity (NetworkX):

quality of partition on each original layer.

Conductance per layer:

fraction of edges leaving communities (lower=better).

Link-prediction AUC:

held-out edges vs non-edges scored by H·H^T dot products.

Stability:

pairwise NMI across restarts; ARI/AMI when ground-truth available.

Consensus:

co-association matrix from multiple runs → Louvain on coassoc → evaluate modularity.

Grid Search & Experimentation Results:

- Grid parameters: k ∈ {10,20,50}, μ ∈ {1e-6,1e-4,1e-2}, normalize_H ∈ {True,False}, init ∈ {louvain,random}, weight_method ∈ {inv_nnz, sqrt_inv_nnz, degree_inv, uniform, custom}
- Use grid_runner.py to iterate experiments, save per-run JSONs and summary JSONs.
- Collect grid_results.csv with summary metrics for each config for easy analysis.
- Consensus pipeline: build co-association, threshold, run Louvain to produce consensus labels.
- High modularity (≈0.7–0.84) suggests strong community structure — we verified with baseline Louvain on each layer.
- High link-AUC (>0.8) indicates H reconstructs pairwise affinities well (but not necessarily semantically meaningful clusters).
- High stability (NMI ≈0.4–0.9 across grids) indicates repeatability; low stability suggests multiple local minima.
- Cross-check modularity vs conductance (low conductance desired).

Practical Issues Observed & Fixes:

1

Memory blowup

when computing dense R = H D H^T for n≈43k (→ ~14GB allocation). Fix: compute objective via traces using H^T H and H^T A H.

2

NetworkX version differences

(from_scipy_sparse_array missing): use custom sparse_to_networkx helper.

3

RuntimeWarning invalid value

in relative change: guard prev_obj finite and handle NaN/inf.

4

Louvain dependency optional:

fallback to random init if python-louvain absent.

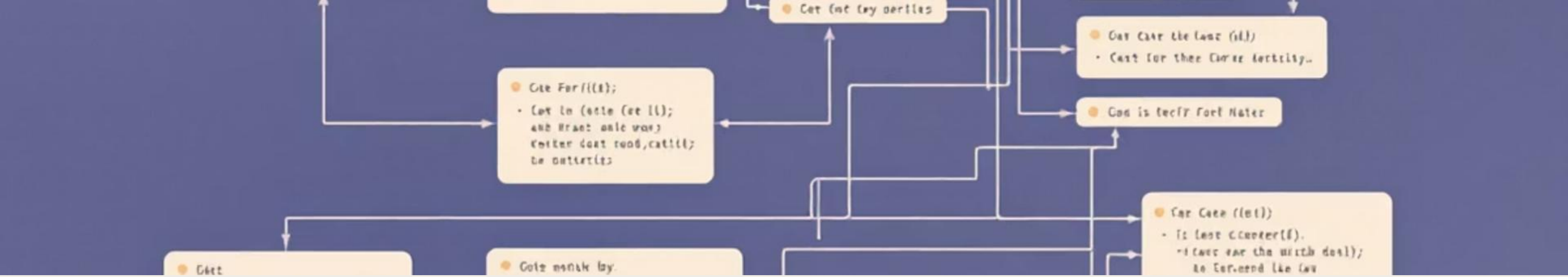
5

Consensus coassoc matrix memory heavy

for large n — implemented sparse coassoc construction with sampling.

Conclusion and Takeaways:

- Implemented and evaluated layer-aware SymNMF with many practical fixes for large graphs.
- Weighting and initialization strongly affect outcomes — inv_nnz useful when layers differ in size.
- Use multiple evaluation metrics (modularity, conductance, AUC, stability) to avoid misleading claims.
- Memory & numerical issues solvable with trace identities, regularization, and sparse consensus building.
- Next steps: learnable layer attention, Leiden init, model selection for k, and improved consensus methods.



Basic Codes and Artifacts

1

align.py — node alignment and .npz generation

2

symm_nmf.py / base.py — model training and evaluation

3

grid_runner.py — run hyperparameter sweep

4

consensus_and_eval.py / consensus_grid.py — coassoc and consensus evaluation

5

results/ — per-run JSONs, summary JSONs, grid_results.csv, consensus JSONs

Thank you for your attention

This project focused on applying **Symmetric Non-negative Matrix Factorization (SymNMF)** for multi-layer community detection across real-world networks.Through preprocessing, alignment, and experimentation with various weight methods and initializations, the model learned shared community structures across layers.Though results were moderate, the framework successfully demonstrated how multi-layer networks can be integrated and analyzed using factorization-based methods.